Subject: FTB workflows
From: bernholdtde@ornl.gov
Date: Tue, 17 Feb 2009 09:23:42 -0500
To: cifts@googlegroups.com
CC: bernholdtde@ornl.gov

Here are five workflows that we came up with...

Requirements:
* Batch manager allows job to continue in face of faults (policy)
* Need standardized nomenclature for system components (including node names) to allow publishers and subscribers to describe (scope) faults in a general way.

Desiderata:
* Components should be able to subscribe with "job scope" to events effecting any node in their allocation and not be bothered by events elsewhere.

Questions:
* What is the proper name for CrayRAS.node_up?
* When reboots happen on a Cray, what is the granularity? Processor? Node? Rack?

IPS (FUSION) SCENARIOS

Background: The Integrated Plasma Simulator (IPS) supports concurrent execution of multiple tasks as part of an overall simulation workflow. A simulation is run as a batch job, allocated a certain number of nodes. The IPS framework incorporates a simple Resource Manager (RM) which is responsible for managing the nodes assigned to the batch job and allocating them to tasks as needed. Individual tasks are launched by the Task Manager (TM) using an 'mpiexec', 'aprun', or similar command, depending on the platform.

WORKFLOW 1: Task dies due to node failure
1) IPS starts
2) IPS subscribes to node_corefail & CrayRAS.node_up events
3) Task t running on node n terminates due to node failure
4) IPS detects unsuccessful completion of task t, wants to determine if error is likely software-related or system-related. TM retains resources for t (protect others from using potentially bad nodes) and waits (with timeout) for FTB to provide additional data
5) RAS detects failure of node n, CrayRAS component sends node_corefail(n) via FTB
6) IPS receives CrayRAS:node_failure(n)
7) RM removes node n from resource pool
8) Event confirms that task t suffered from a system problem, and should be re-run. (If the node_corefail event was not received in time, conclusion would have been that it was a software error, in which case you would probably abort the simulation.)
9) TM releases original allocation for task t
10) TM requests new allocation (which will avoid using the down node),

and launches task again.

11) Time passes, simulation continues...
12) Node is rebooted and returned to service, CrayRAS component sends CrayRAS.node_up(n) via FTB
13) IPS receives CrayRAS.node_up(n)
14) RM restores node n to resource pool. Node n will be used in subsequent allocations
15) Simulation completes
16) IPS unsubscribes and finalizes FTB
17) IPS terminates

WORKFLOW 2: Task hangs due to node failure

1) IPS starts
2) IPS subscribes to node_corefail & CrayRAS.node_up events
3) Task t running on node n hangs due to node failure
4) RAS detects failure of node n, CrayRAS component sends node_corefail(n) via FTB
5) IPS receives CrayRAS:node_failure(n)
6) RM removes node n from resource pool
7) TM observes that task t, which uses node n, has not terminated. Kills task t
8) When task t terminates, TM releases original allocation for task t
9) TM requests new allocation (which will avoid using the down node), and launches task again. (TM should have a limit on retries before giving up.)
10) Time passes, simulation continues...
11) Node is rebooted and returned to service, CrayRAS component sends CrayRAS.node_up(n) via FTB
12) IPS receives CrayRAS.node_up(n)
13) RM restores node n to resource pool. Node n will be used in subsequent allocations
14) Simulation completes
15) IPS unsubscribes and finalizes FTB
16) IPS terminates

Possible extensions and generalizations:

* Any events logically equivalent to node_failure/node_up in their effect on the running task could be used
* If application sees a rapid sequence of unsuccessful completions when trying to restart a job it could publish an event indicating suspicious behavior of the nodes involved. Need a "diagnostic" component to be listening and try to correlate with other indicators.

MOLECULAR DYNAMICS SCENARIOS

WORKFLOW 3: MD code using fault-tolerant MPI implementation

Assumptions:

* Using an MPI implementation which it capable of continuing with failed nodes, as long as no other node tries to communicate with it (leaving a hole in the communicator). This is one of the modes of the UTK FT-MPI implementation, for example.
* That neighbor state information, typically communicated among the nodes of an MD code, provides sufficient redundancy to reconstruct

the full state in the event of a node failure.

1) MD starts
2) MD subscribes to node_corefail & CrayRAS.node_up events
3) RAS detects failure of node n, CrayRAS component sends node_corefail(n) via FTB
4) MD receives node_corefail(n)
5) MD redistributes local and neighbor data to form a complete state for N-1 nodes
6) MD repeats previous step with newly redistributed data, continues on
7) Time passes and simulation continues...
8) Node is rebooted and returned to service, CrayRAS component sends CrayRAS.node_up(n) via FTB
9) MD receives CrayRAS.node_up(n)
10) MD redistributes local and neighbor data to form a complete state for N nodes
11) MD continues with next step
12) Simulation completes
13) MD unsubscribes and finalizes FTB
14) MD terminates

WORKFLOW 4: MD code using fault-intolerant MPI implementation

Assumptions:
* Using an MPI implementation which aborts in the event of node failures
* MD code uses disk-based checkpoint/restart
* MD code can restart from a checkpoint taken on a different number of nodes

Notes:
* The fault tolerance in this scenario is not implemented in the application itself (except from the basic checkpoint/restart support), but rather in a driver of some kind, which is interfaced with the FTB. For example, a Python driver script with Aniruddha's Python FTB binding.
* For simplicity, we assume that the system removes failed nodes from the pool and will not try to launch jobs on them. If this is not the case, the driver could be extended to include a simple local resource manager to track failed nodes and avoid using them.

1) Driver starts
2) Driver subscribes to node_corefail & CrayRAS.node_up events
3) Driver starts MD code
4) MD code terminates due to node failure
5) RAS detects failure of node n, CrayRAS component sends node_corefail(n) via FTB
6) Driver receives node_corefail(n)
7) Driver adjusts MD configuration to use N-1 nodes
8) Driver relaunches MD
9) Repeat until MD terminates successfully or number of nodes gets too small.
10) Driver unsubscribes and finalizes FTB
11) Driver terminates

Possible extensions and generalizations:
* Any application with similar FT behavior/requirements could be

substituted for MD

COMMUNICATIONS LINK FAILURE SCENARIO

Notes:
* On the Cray XT, link failures are common. They can be routed around at the system level, but only at boot time. In this scenario, we propose that the MPI layer should be able to dynamically reroute around such failures.
* For demonstration purposes, could be done in MPICH or OpenMPI rather than Cray's proprietary MPI

WORKFLOW 5: Dynamic rerouting around link failures
1) MPI-based application starts
2) MPI subscribes to link_inactive
3) RAS detects failure of communication link l, CrayRAS component sends link_inactive(l) via FTB
4) MPI receives link_inactive(l)
5) MPI removes link from its routing tables
6) Generate NAKs for all senders who had unacknowledged messages in flight over link l, triggering a resend
7) Application continues
8) Application calls MPI_finalize
9) MPI unsubscribes and finalizes FTB
10) Application completes

Possible extensions and generalizations:
* Might also be done with other interconnects and MPI implementations, for example IB and MVAPICH

ADDITIONAL SCENARIO IDEAS

Policy interactions: can tell scheduler that job can use additional nodes. Scheduler sends additional_nodes_available events, treated similarly to node_up

Proactive process migration

Scheduler sends events for time remaining, application uses to checkpoint and terminate cleanly rather than being killed at time limit.

If need to reboot node, get advance notification events to allow checkpoint, clean termination

```
--
David E. Bernholdt                   |   Email: bernholdtde@ornl.gov
Oak Ridge National Laboratory        |   Phone: +1 (865) 574 3147
http://www.csm.ornl.gov/~bernhold/   |   Fax:   +1 (865) 576 5491
```

--~--~---------~--~----~------------~-------~--~----~

-~----------~----~----~----~------~----~------~--~---